AD-A069 094 MASSACHUSETTS INST OF TECH LEXINGTON LINCOLN LAB F/G 9/2
TWO WAYS TO INTERFACE A MOTOROLA M6800 MICROPROCESSOR TO A MODC--ETC(U)
FEB 79 L E EATON F19628-78-C-0002
UNCLASSIFIED ESD-TR-79-29 NL

1 OF 1
AD
A069 094

END
DATE
FILMED
7--79
DDC

12

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

LINCOLN LABORATORY

TWO WAYS TO INTERFACE A MOTOROLA M6800 MICROPROCESSOR TO A MODCOMP COMPUTER

L. E. EATON

Group 94

PROJECT REPORT ETS-34

22 FEBRUARY 1979

LEXINGTON MASSACHUSETTS

ABSTRACT

Two interfaces have been developed between a Motorola M6800 microprocessor and a ModComp computer. One interface uses a word by word handshaking-interrupt design to control the main telescope at the GEODSS ETS. The other interface uses a stringed message transfer for communication to the microprocessor development system. Both interfaces are described in this report.

TABLE OF CONTENTS

I. INTRODUCTION

Two interfaces have been developed at the GEODSS ETS between a Motorola M6800 microprocessor and a ModComp computer. One interface is a "word by word handshaking design", and the other is a "stringed message" design.

The stringed message interface is used at GEODSS as the communication between the ModComp and the microprocessor that controls a 31" Boller and Chivens telescope. The handshaking design interfaces the ModComp to Motorola's EXORciser (Motorola's development system for their microprocessor).

The two interfaces incorporate two philosophies of design. The handshaking interface allows one word at a time to be transferred between the two systems. A status bit at the microprocessor and an interrupt at the ModComp implement the handshaking. The stringed message interface allows the sending device to transmit up to 16, 16-bit words before the receiving device is required to read the data.

The body of this report is divided into five sections. The first (Section II) is a brief description of Motorola's Peripheral Interface Adapter (PIA) and the mode in which it is used at the ETS. The PIA is Motorola's method of interfacing the microprocessor's I/O bus to the outside world.

The next section (Section III) describes how the PIA's, the external telescope control logic, and the I/O software interact. This interaction is based on the development of an external address and data bus implemented through the PIA's.

This separate external address and bi-directional data bus (separate from the microprocessor's I/O bus) was developed, using the PIA, to allow the logic to appear to the software as 256 external 8-bit registers. These registers were used in the design of the controlling logic for a 31" Boller and Chivens telescope as well as the communications to a ModComp computer.

Section IV is the description of the word by word interface between the ModComp and the microprocessor. The word by word interface is used for communication between the EXORciser and the ModComp. This interface is used to allow the ModComp to be a replacement for the original teletype that was the EXORciser's line printer and data storage medium.

The stringed message interface (described in Section V) is used at the ETS to transmit information between the ModComp and the Motorola microprocessor controlling the 31" Boller and Chivens telescope. This interface is described as a message type interface because as many as 16, 16-bit words can be transmitted before the receiving unit must read the message. The interface allows the ModComp to send data in 16-bit words while the microprocessor reads the data in 8-bit bytes. In sending data from the microprocessor to the ModComp, the interface packs the 8-bit microprocessor words into 16-bit words for the ModComp to read.

Section VI describes the software considerations that would be used by a programmer to implement these interfaces. No attempt is made to explain the actual software handlers used.

This report attempts to meet two goals. One is to give the reader, who is interested in an interface between a ModComp computer and a Motorola M6800 microprocessor, an overview of two specific types of interfaces. Secondly, it is hoped that this report describes the interfaces in enough detail so that the reader obtains a basic understanding of the design at the ETS. Although the reader will probably need logic prints, their inclusion in this report would be cumbersome. Consequently, block diagrams and verbal descriptions have been used as substitutes for the prints. Also, each section refers to the appropriate logic prints that are filed at the GEODSS ETS site.

II. A BRIEF EXPLANATION OF MOTOROLA'S PERIPHERAL INTERFACE ADAPTER

A. IN GENERAL

Motorola's Peripheral Interface Adapter (PIA), MC6820, is an IC that interfaces the microprocessors I/O bus system to the outside world. Although a very detailed description of the chip is given in the Motorola Application Manual, a brief description of its functions is given here, based on how the PIA is used in our system.

Motorola's microprocessor system does not have any I/O instructions. Instead, all peripheral IC's, such as a PIA, have a unique location in core.

Each PIA has two 8-bit peripheral register channels with two associated 8-bit control registers, so that a PIA appears as 4 memory locations in core. Also accessible within the PIA is a Data Direction Register (DDR). This register is used to define the 8-bit data lines as either inputs or outputs. There are also two control lines for each channel on the PIA. The A (B) channel uses CA1 (CB1) and CA2 (CB2). The PIA's configuration prejudices the user to define channel A as an input channel and B as an output channel. They are used this way in the GEODSS system.

The PIA's are used in both the EXORciser interface design and the telescope interface design to perform four functions:

1. Input data from the external logic
2. Output data to the logic

3. Input a 3-bit priority interrupt number

4. Output an 8-bit logic register address

These functions need two input channels and two output channels and thus two PIA's were used.

Table 1 gives the core locations where these channels are configured in the ETS system, along with the software mnemonic used in the telescope control program. It is advantageous to use a mnemonic to allow the address of each PIA to be defined only once in the program. Figure 1 is a block diagram of the PIA illustrating the above.

TABLE 1

Address, Software Mnemonic, and Register

Description of System PIA's

ADDRESS	SOFTWARE MNEMONIC	REGISTER
$7FF8	INDATA	PIA #1, Chan A periph reg or DDR
$7FF9	PIACIA	PIA #1, Chan A ctrl reg
$7FFA	OUDATA	PIA #1, Chan B periph reg or DDR
$7FFB	PIAC1B	PIA #1, Chan B ctrl reg
$7FFC	INTREG	PIA #2, Chan A periph reg or DDR
$7FFD	PIAC2A	PIA #2, Chan A ctrl reg
$7FFE	ADRBUS	PIA #2, Chan B periph reg or DDR
$7FFF	PIAC2B	PIA #2, Chan B ctrl reg

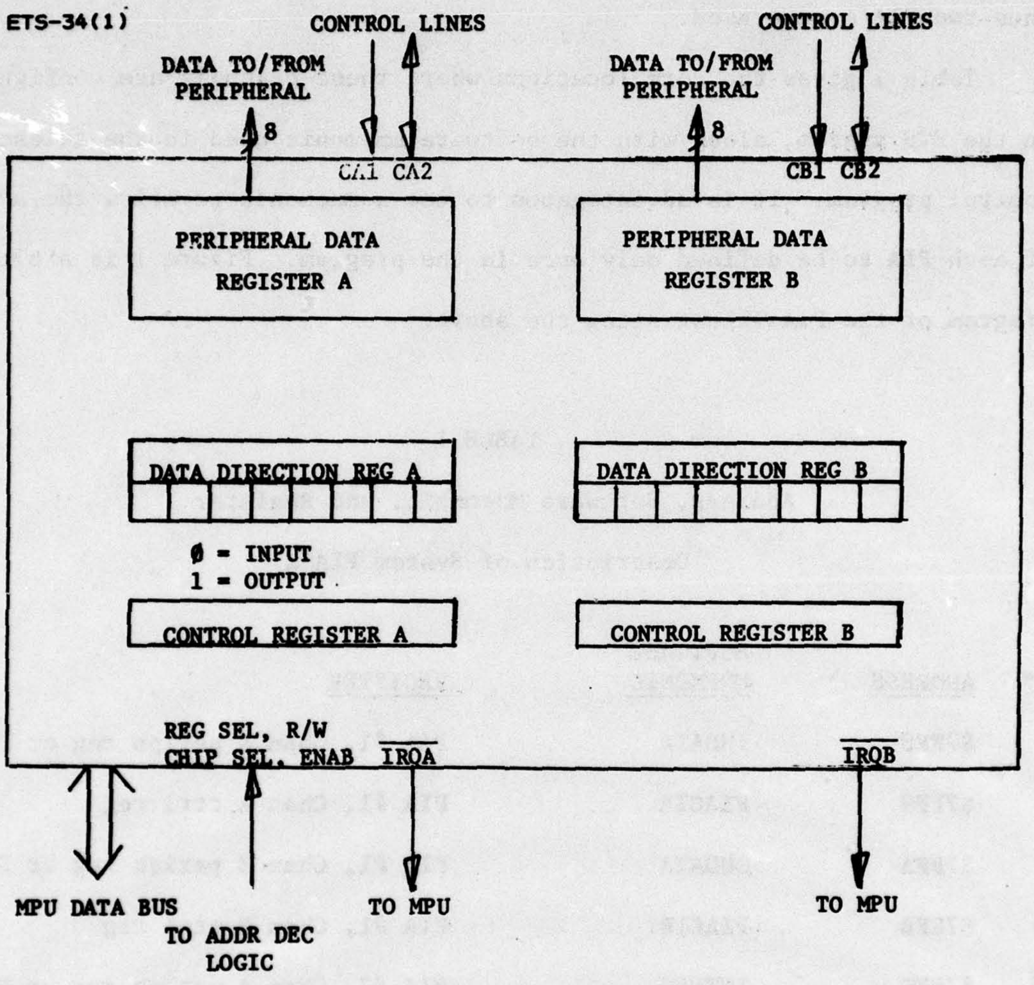


Fig. 1. Block diagram of Peripheral Interface Adapter (PIA).

B. THREE REGISTERS WITHIN EACH CHANNEL OF THE PIA

1. Peripheral Register - This is the register that either inputs or outputs data (depending on the state of the DDR) from/to the outside world.

2. Data Direction Register (DDR) - This register defines the direction of each of the 8 peripheral register lines as being input or output lines (1 = output, Ø = input).

3. Control Register - This register is used to control the various functions of the PIA. It allows the software to do the following:

   a) enable/disable interrupts within the PIA

   b) access the DDR

   c) determine if an interrupt has occurred on a PIA

   d) define the state of one of the hardware control lines to be in the pulse mode, bit 3 following mode, or handshaking mode

   e) set the other control line so that an interrupt occurs on a low-high or high-low going edge.

As a note - each half of a PIA looks like two core locations to the software. But there are 3 registers that can be accessed by the software. The peripheral register and the DDR have the same address. The control register has the other address. Whether the peripheral register or the DDR is being accessed is determined by the state of bit Ø2 in the control register. If -

bit Ø2 = Ø : software can talk to DDR

= 1 : software can talk to peripheral register.

C. THE TWO CONTROL LINES OF EACH CHANNEL

Each channel of the PIA has two control lines CA1 (CB1) and CA2 (CB2). These lines are used for various modes of interfacing to a peripheral device. The particular mode is determined by the control register. For our application, the PIA's are configured in the so-called pulse mode of operation. That is, on control line 2 a pulse accompanies the data upon output to channel B (CB2). A pulse also occurs when data are input from channel A (CA2). Thus, the external logic has a pulse associated with the data during I/O operations. The logic calls this pulse a data sync pulse (DSP).

The other control line is configured as an interrupt control. For our system, it is only used in the one PIA that is interfaced to the priority interrupt logic. This interrupt is simply passed through the PIA to the $\overline{\text{IRQ}}$ input of the microprocessor.

D. CONFIGURING THE CONTROL REGISTERS

As previously stated, PIA's can be configured in several modes, all of which are described in detail in the Application Manual and the Microprocessor Course Manual. However, only the pulse mode will be described here. This mode sends a pulse on control line CA2 (or CB2) when the software does a read (or write) to the particular PIA. This pulse indicates to the logic that the data is valid (for OUTDATA and ARDBUS) or that the data is being read (for INDATA and INTREG). For this configuration, the control register bits 3 - 5 are set as follows:

bit 5 = 1

bit 4 = Ø

bit 3 = 1

Control lines CA1 and CB1 are used as interrupt lines. For an interrupt to occur on the low to high transition of a line, bit 1 of the control register must be 1. Enabling of the interrupt is controlled by bit Ø.

bit 1	bit Ø	
1	Ø	low to high transition (bit 1 = 1) of CA1 (CB1) interrupt disabled (bit Ø = Ø)
1	1	low to high transition (bit 1 = 1) of CA1 (CB1) interrupts enabled (bit Ø = 1)

The ETS system uses control line CA1 on PIA #2 as the only interrupt to the microprocessor CPU. Bit 7 of the PIA's control register will be set to 1 when an interrupt occurs. This is passed on to the MPU's $\overline{\text{IRQ}}$ line causing it to go low. The software's interrupt handler must then read the data from the PIA (INTREG) to determine which level caused the interrupt. This read will clear bit 7, allowing another interrupt sequence to occur.

The Data Direction Register (DDR) is accessed by setting bit 2 in the control register to Ø. Then, instead of addressing the peripheral register, the DDR will be addressed. Setting all 1's in the DDR defines the peripheral register as an output register and all Ø's defines it as an input register.

The following initialization program will put the PIA's in the proper configuration for the ETS hardware.

```
                 ORG $FFF8                        LOC OF PIA'S

        INDATA   RMB 1                            PIA1A per reg or DDR ($7FF8)
        PIAC1A   RMB 1                            PIA1A ctrl reg ($7FF9)
        OUDATA   RMB 1                            PIA1B per reg or DDR ($7FFA)
        PIAC1B   RMB 1                            PIA1B ctrl reg ($7FFB)
        INTREG   RMB 1                            PIA2A per reg or DDR ($7FFC)
        PIAC2A   RMB 1                            PIA2A ctrl reg ($7FFD)
        ADRBUS   RMB 1                            PIA2B per reg or DDR ($7FFE)
        PIAC2B   RMB 1                            PIA2B ctrl reg ($7FFF)
                 ORG       $XXXX
                 LDAA      #$Ø                    select DDR for all PIA's
                 STAA      PIAC1A
                 STAA      PIAC1B
                 STAA      PIAC2A
                 STAA      PIAC2B
                 LDAA      #$FF                   define DDR as output
                 STAA      ADRBUS                 for ADRBUS and OUDATA
                 STAA      OUDATA
                 LDAA      #$ØØ                   define DDR as input
                 STAA      INDATA                 for INDATA and INTREG
                 STAA      INTREG
                 LDAA      PRDIMPM                define all PIA's as -
                 STAA      PIAC1A                 sel per reg (PR), disable int
                 STAA      PIAC1B                 (DI) and pulse mode (PM)
                 STAA      PIAC2A
                 STAA      PIAC2B
                 END
        PRDIPM   FCB       $2E                    Sel periph reg (PR), dis int (DI),
                                                    Pulse Mode (PM)
        PREIPM   FCB       $2F                    Sel periph reg (PR), en int (EI),
                                                    Pulse Mode (PM)
```

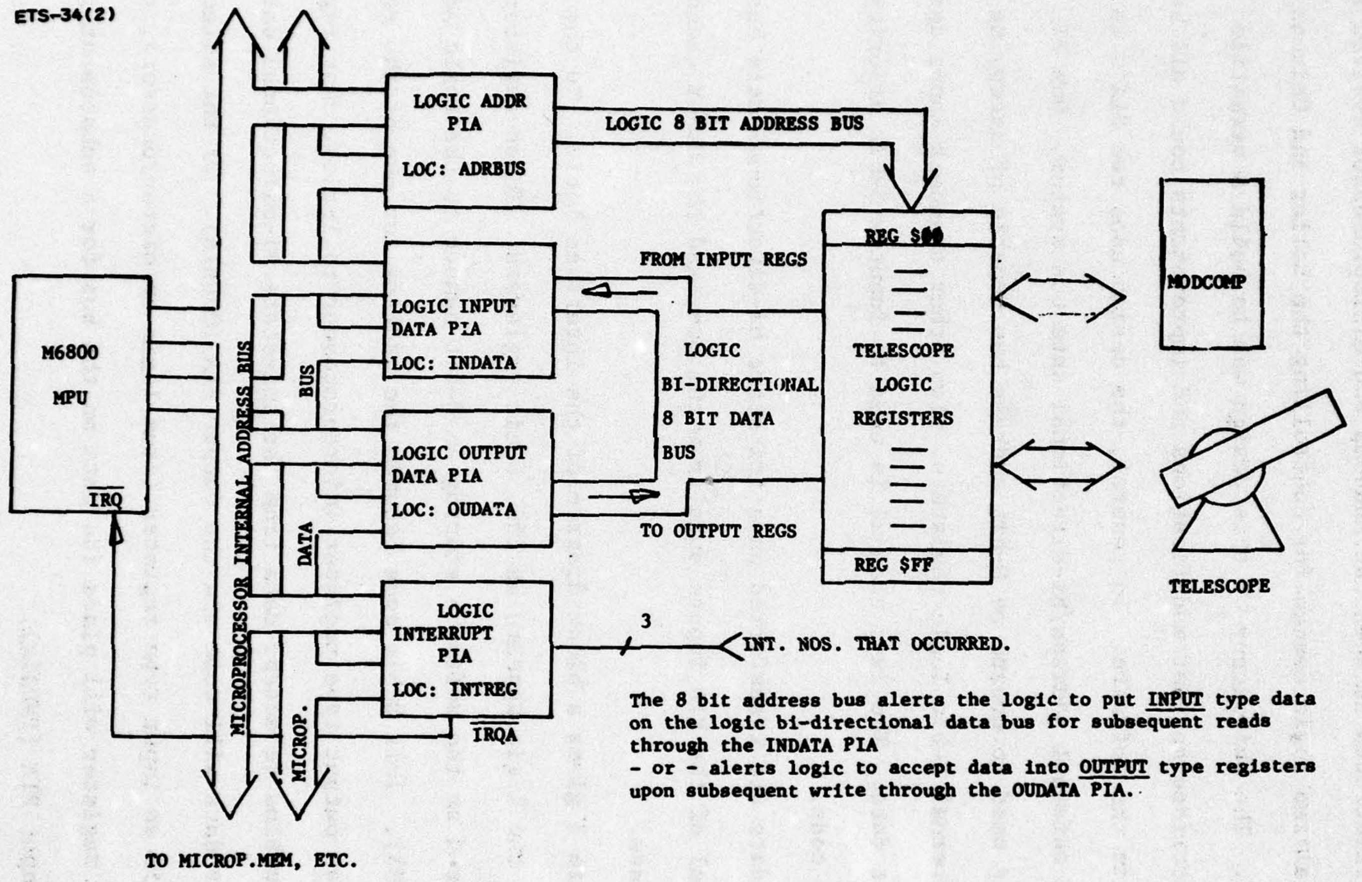
III. THE TELESCOPE CONTROL LOGIC ADDRESS BUS, BI-DIRECTIONAL DATA BUS AND PRIORITY INTERRUPT STRUCTURE

The interfaces between the Modcomp and microprocessor evolved from the generalized logic design for controlling the Boller and Chivens 31" telescope. The philosophy of this design was to build a versatile enough interface so that modifications and improvements could all be done within the software. In general, the design uses two PIA's to create an external address/bi-directional data bus system. One PIA channel is used to create an 8-bit address bus capable of accessing up to 256 external 8-bit logic registers. Two other channels input data and output data. The last channel is used to input a 3-bit priority interrupt code.

The data are transferred on a tri-state bi-directional data bus. One channel of the PIA inputs data from the bus, and the other channel outputs data.

Figure 2 gives a block diagram of the interface logic. To the software, the logic appears as 256, 8-bit registers. These registers are accessed by the software storing an 8-bit number to the Logic Address PIA (ADRBUS). This 8-bit code becomes the address for one of the registers. If it is an output type register (microprocessor to logic), that register will be enabled to accept data from the tri-state bi-directional data bus. This data will come via the Output PIA (OUDATA). If the selected register is an input type register (from logic to microprocessor), then only that register will place its data on the bus for a subsequent read via the Input PIA (INDATA).

ETS-34(2)



The 8 bit address bus alerts the logic to put INPUT type data on the logic bi-directional data bus for subsequent reads through the INDATA PIA
- or - alerts logic to accept data into OUTPUT type registers upon subsequent write through the OUDATA PIA.

Fig. 2. Block diagram of telescope interface logic to/from microprocessor.

Each time a word is input or output, the address bus is incremented by 1. This allows successive registers to be accessed without sending a new address to the ADRBUS PIA. This increases I/O speed considerably.

The priority interrupt structure is shown in Figure 3. The structure interfaces an 8 level priority scheme to the MPU $\overline{\text{IRQ}}$ level. One channel of the PIA is dedicated to these interrupts.

The software can be interrupted on any one of 8 levels (level Ø being the highest priority). These levels are decoded into a 3-bit code. The interrupt is sent to the PIA (INTREG) and then transferred on to the MPU, $\overline{\text{IRQ}}$ level. The software then inputs the 3-bit code from the interrupt PIA to determine which level interrupted the system.

These interrupts are enabled and disabled by the software storing data into the enable register which is one of 256 registers in the system.

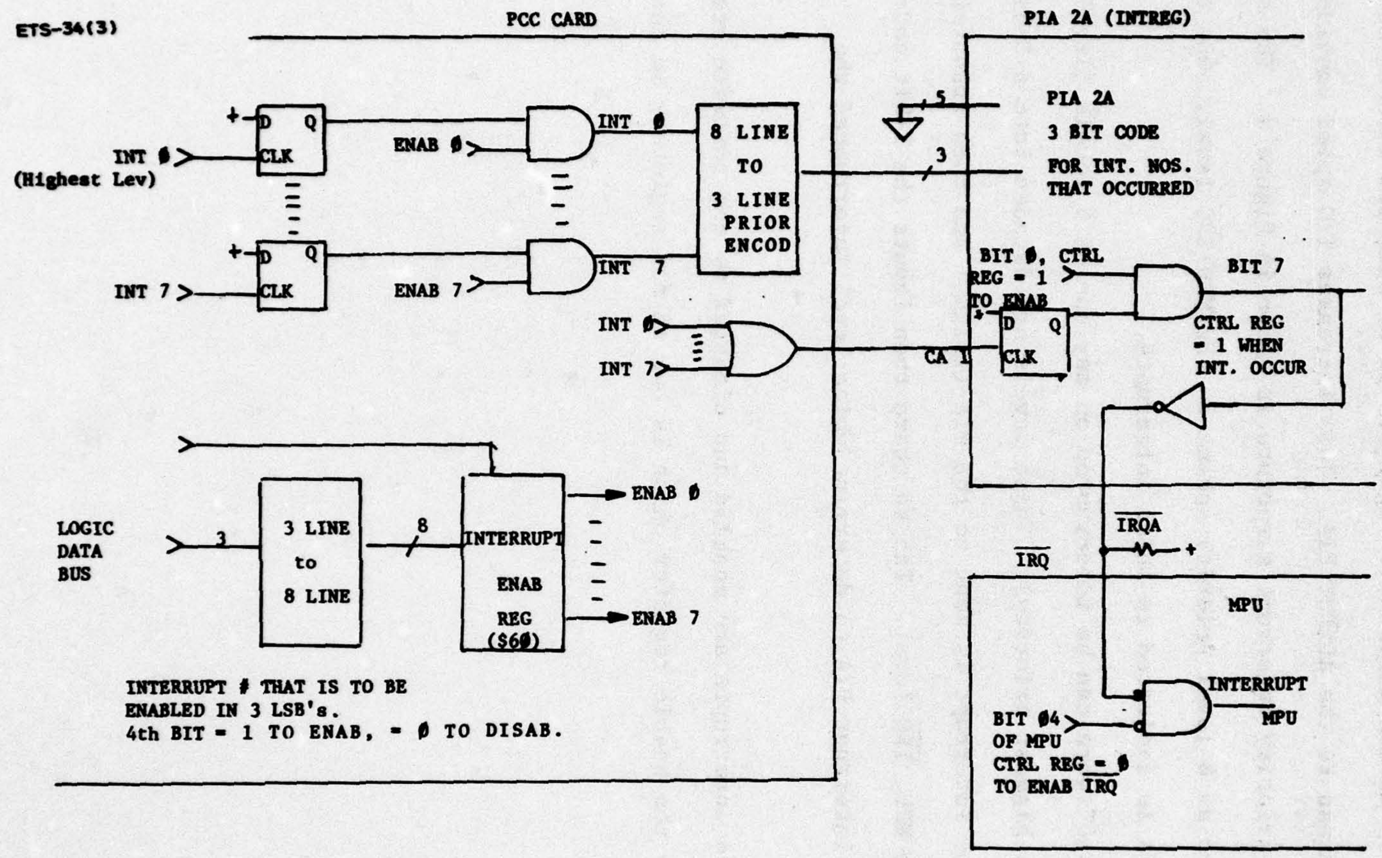

ETS-34(3)
PCC CARD
PIA 2A (INTREG)
INT ø (Highest Lev)
INT 7
D Q CLK
ENAB ø
ENAB 7
INT ø
INT 7
8 LINE TO 3 LINE PRIOR ENCOD
5
3
PIA 2A
3 BIT CODE
FOR INT. NOS. THAT OCCURRED
BIT ø, CTRL REG = 1 TO ENAB
BIT 7
CTRL REG = 1 WHEN INT. OCCUR
CA 1
LOGIC DATA BUS
3 LINE to 8 LINE
8
INTERRUPT ENAB REG ($6ø)
INTERRUPT # THAT IS TO BE ENABLED IN 3 LSB's.
4th BIT = 1 TO ENAB, = ø TO DISAB.
IRQA
IRQ
MPU
INTERRUPT MPU
BIT ø4 OF MPU CTRL REG = ø TO ENAB IRQ

Fig. 3. Block diagram of 8 bit priority interrupt design.

IV. WORD BY WORD HANDSHAKING INTERFACE

This interface is a one word (8 bits) type transfer between the ModComp and the microprocessor. The handshaking is done by an interrupt at the ModComp and by status checking at the microprocessor. Figure 4 shows the block diagram.

The register type interface to the microprocessor described in Section III, is used to define an input register, an output register and a status register for the data transfer. The word by word interface was incorporated to allow the EXORciser to use the ModComp as a peripheral device. The EXORciser receives source code, etc., from the ModComp and sends object code and listing information to the ModComp. A slight modification to the Motorola EXBUG I/O software was made to adapt this interface.

The ModComp computer interfaces to the outside world through a system called an Input/Output Interface Subsystem (I/OIS). This is a very straight forward design. Each I/OIS channel can be configured as an input (16 bits) or output (16 bits) with an associated sync pulse. There is also an interrupt coupler card which allows external interrupts to be coupled to the ModComp.

For the word by word handshaking interface one 16-bit channel is used for an input channel and one as an output. Only the lower 8 bits are used since this is the word size for the microprocessor.

I/OIS channel 11 is used to send data from the ModComp to the EXORciser. This is an output channel. The ModComp sends 8 bits of data (8 LSB's) when it is interrupted on I/OIS Data Interrupt 1. The vectoring

ETS-34(4)

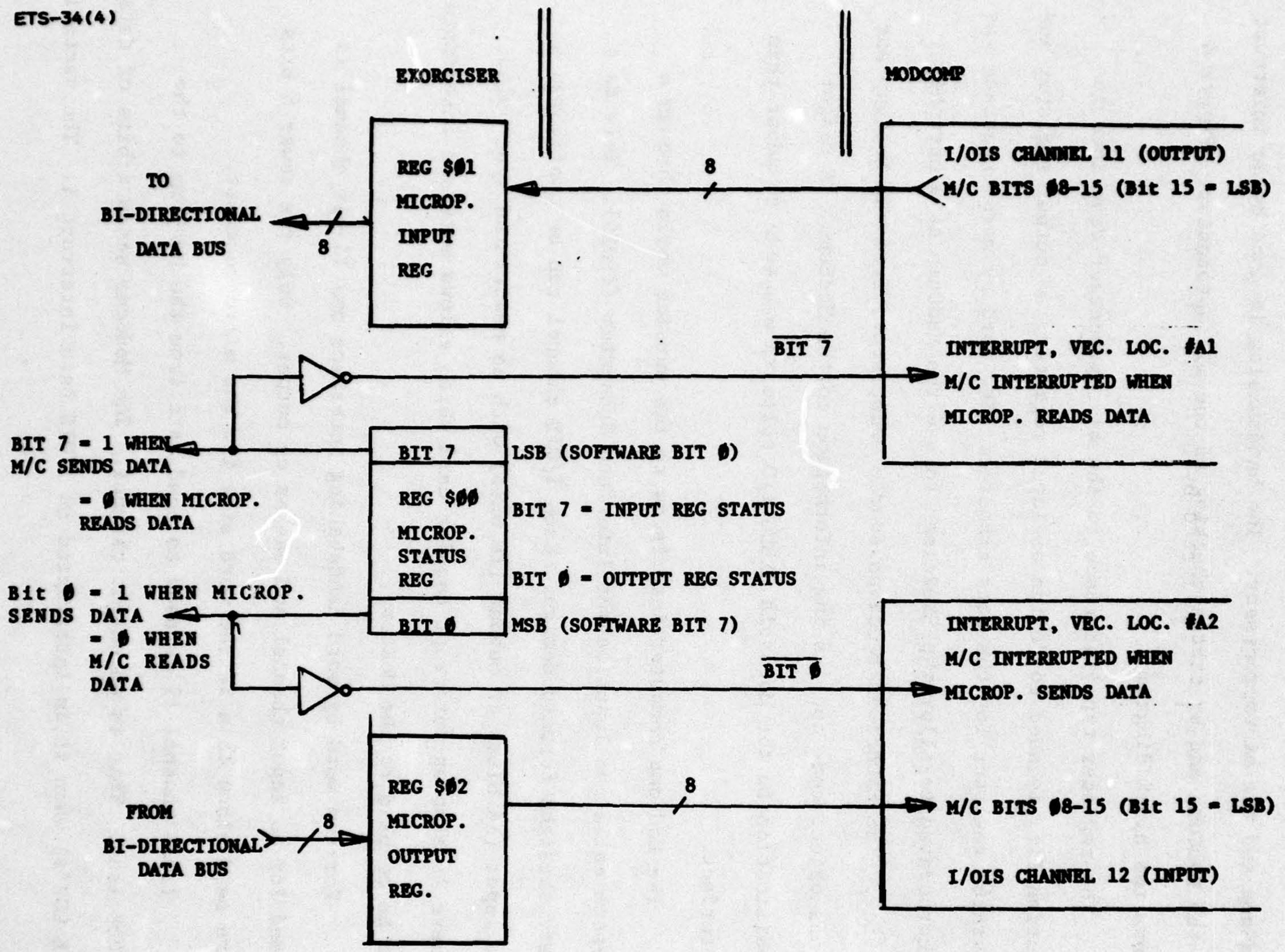


Fig. 4. Block diagram of ModComp to Motorola EXORciser interface.

location within the ModComp is core location #A1 (# = hex). The interrupt comes when the microprocessor reads the last word sent.

When the ModComp is to receive data, it does so through channel 12 (an input channel) and is alerted by receiving an interrupt on Data Interrupt 2. The ModComp vectoring location is found at #A2. The interrupt occurs when the microprocessor sends a word.

There are three of the 256 external registers used at the microprocessor as the transfering registers. Register #Ø1 is used to read data from the ModComp, and register #Ø2 is used to send data to the ModComp. Register #ØØ is used as a status register to tell the software when to send or receive data.

The microprocessor can read data (from reg #Ø1) whenever the LSB in register #ØØ is a 1 (status is good), indicating the ModComp has sent a word. By reading the data, the LSB is changed to a zero (status bad). The bit goes good again when the ModComp sends another word.

The microprocessor may send a word to the ModComp whenever the MSB in register ØØ is a 1 (status good). Upon sending a word to register Ø2, the ModComp is interrupted, and the status bit is set to a Ø. When the ModComp reads the word, the status goes good (i.e., MSB reg #ØØ = 1), and the microprocessor can send another word.

The interrupt/status arrangement is an invert type interface. That is, when the microprocessor status is good it may send data. When it does send data, the status goes bad and the ModComp is interrupted.

The logic prints for this interface can be found on prints titled, EXI, AICN-11 and AICN-12.

V. STRINGED MESSAGE INTERFACE

This interface is used to send telescope data to the ModComp and to receive commands from the ModComp.

The same address/register interface is used at the microprocessor as described in Section III and the I/OIS on the ModComp is used for data input and output (one channel for each).

The basic design incorporates a scratch pad memory (SPM). For each direction up to 16, 16-bit words can be stored in a SPM before the receiving device reads the data. These two 16 word by 16 bit SPM's appear to the microprocessor software as 32 input type registers and 32 output type registers of the 256 total registers described in Section III.

The logic diagrams for this interface can be found on the prints titled AICN-8B, AICN-9B, TIB-4 and PCC (TIB-6).

A. DATA FROM MICROPROCESSOR TO MODCOMP VIA A SPM

Figure 5 shows a block diagram of the transfer of data from the microprocessor to the SPM and then on to the ModComp. The SPM is the central tie point. The design packs the data from 8-bit words out of the microprocessor to 16-bit words into the ModComp.

1. <u>Output from the Microprocessor to the SPM</u>

Since the SPM is 16 words x 16 bits long, and the microprocessor is an 8-bit word, the SPM appears to the microprocessor as 32 hardware output type registers. The starting address is $EØ (end = $FF).

Since the ModComp has to read the same memory, to avoid conflict it was decided to let the microprocessor have control of the memory at

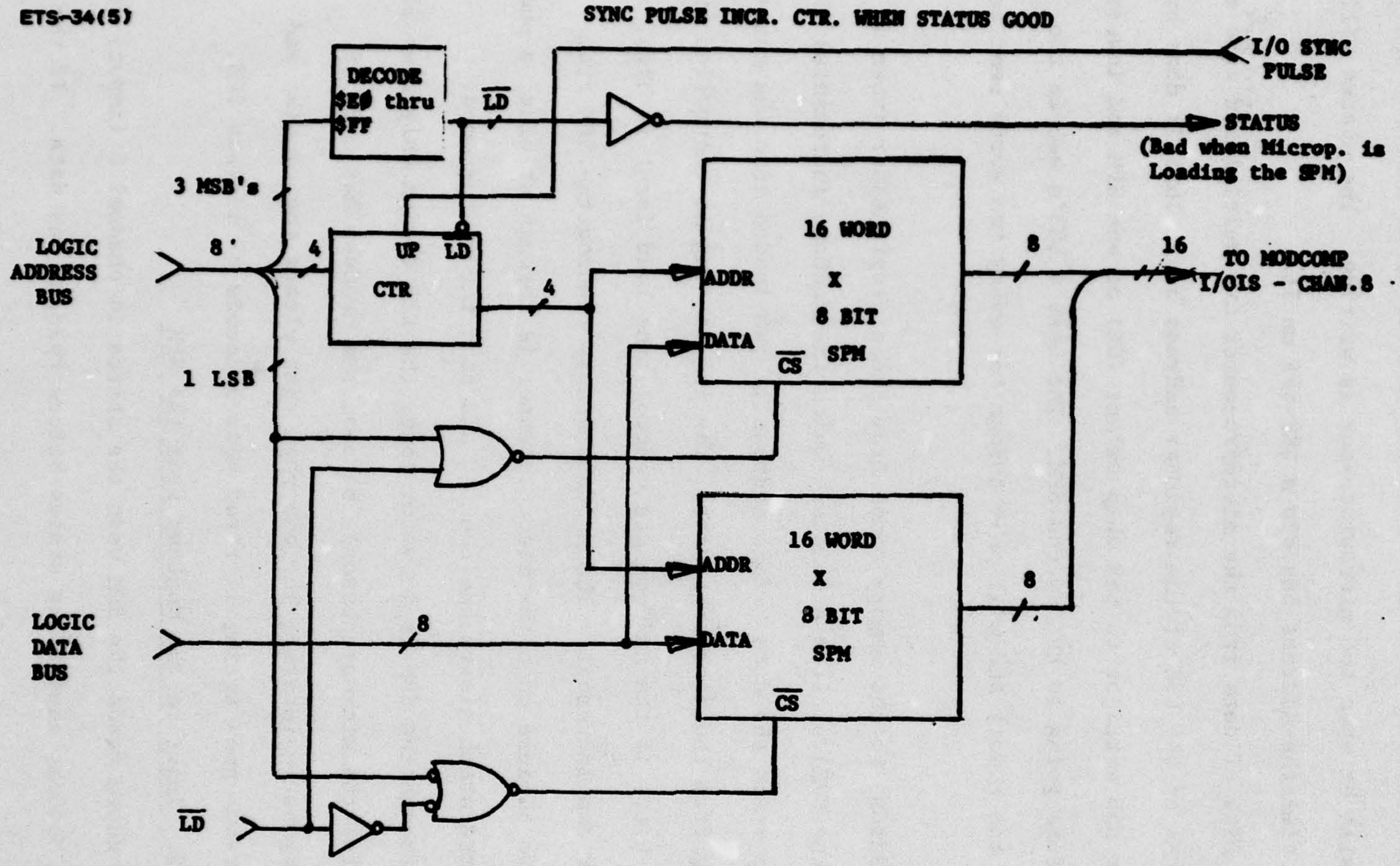


Fig. 5. Block diagram of microprocessor to ModComp — stringed message transfer — interface (AICN-8B & TIB-4).

any time. The ModComp will not read when the status to the ModComp is bad which will be when the microprocessor is writing. The status will be bad any time the address bus has a $EØ-$FF on it.

The 8 bits of data from the microprocessor are multiplexed into the 2, 8 bit SPMs by the LSB of the register address bus. This is done by putting the LSB straight to the chip select (CS) of one SPM and inverting the LSB before going to the second SPM. The next 4 LSB's become the address to the memory and will only change for every two words sent to the SPM's.

The address to the memory comes from the microprocessor (when it writes to the SPM) or from a counter (which the ModComp increments) when the ModComp reads the data. This address is jam loaded into the counter when coming from the microprocessor. The jam loading is controlled by the status bit. If the ModComp has control, the load input is high, and the counter can increment. The ModComp starts reading the SPM from address zero because when the microprocessor is through writing, a one-shot is fired which clears the counter and sets the status good.

Even though the SPM is 16 words long, the ETS design only uses 15 words (30 for the microprocessor), because the ModComp has only 15 general purpose registers. The one-shot that clears the counter and sets the status good again, is fired upon a decode of address $FE.

2. <u>Input to the ModComp from the SPM</u>

The ModComp reads the SPM when the status to channel 8 (input) is good. The ModComp checks the status before reading the data. If the status is good, it proceeds to read the 15 words. The status is also checked after the data has been read by the ModComp. If the status is bad after the transfer, the ModComp ignores the data, waits for the status to go good, then rereads the SPM. The ModComp will read the SPM in the burst mode (15 successive reads, non-interrupted). This takes approximately 22 µs. The microprocessor takes longer to write new data and thus a status check after the burst mode read will indicate if the microprocessor is writing data.

The SPM address counter will always be left cleared by the microprocessor. Therefore, the ModComp knows that the first word read is in location Ø. When the counter gets to 15, it is automatically cleared back to Ø. It is incremented by one for each read.

B. DATA FROM MODCOMP TO MICROPROCESSOR VIA SPM

Figure 6 shows the block diagram of the transfer of data from the ModComp to another SPM and then to the microprocessor. The SPM is again the main tie point and allows the data to go from 16-bit words from the ModComp to 8-bit words into the microprocessor.

1. Data from the ModComp to the SPM

For this direction, the ModComp is allowed to send data if the status to I/OIS channel 9 is good. This will be true if the microprocessor has cleared the status (interrupt) flip-flop.

The address to the SPM is multiplexed by the decoding of the address bus for addresses \$40-\$5F. These are the 32, 8-bit registers that the SPM represents. If the microprocessor is not selecting these registers, then the multiplexer will allow a counter to address the memory. This counter

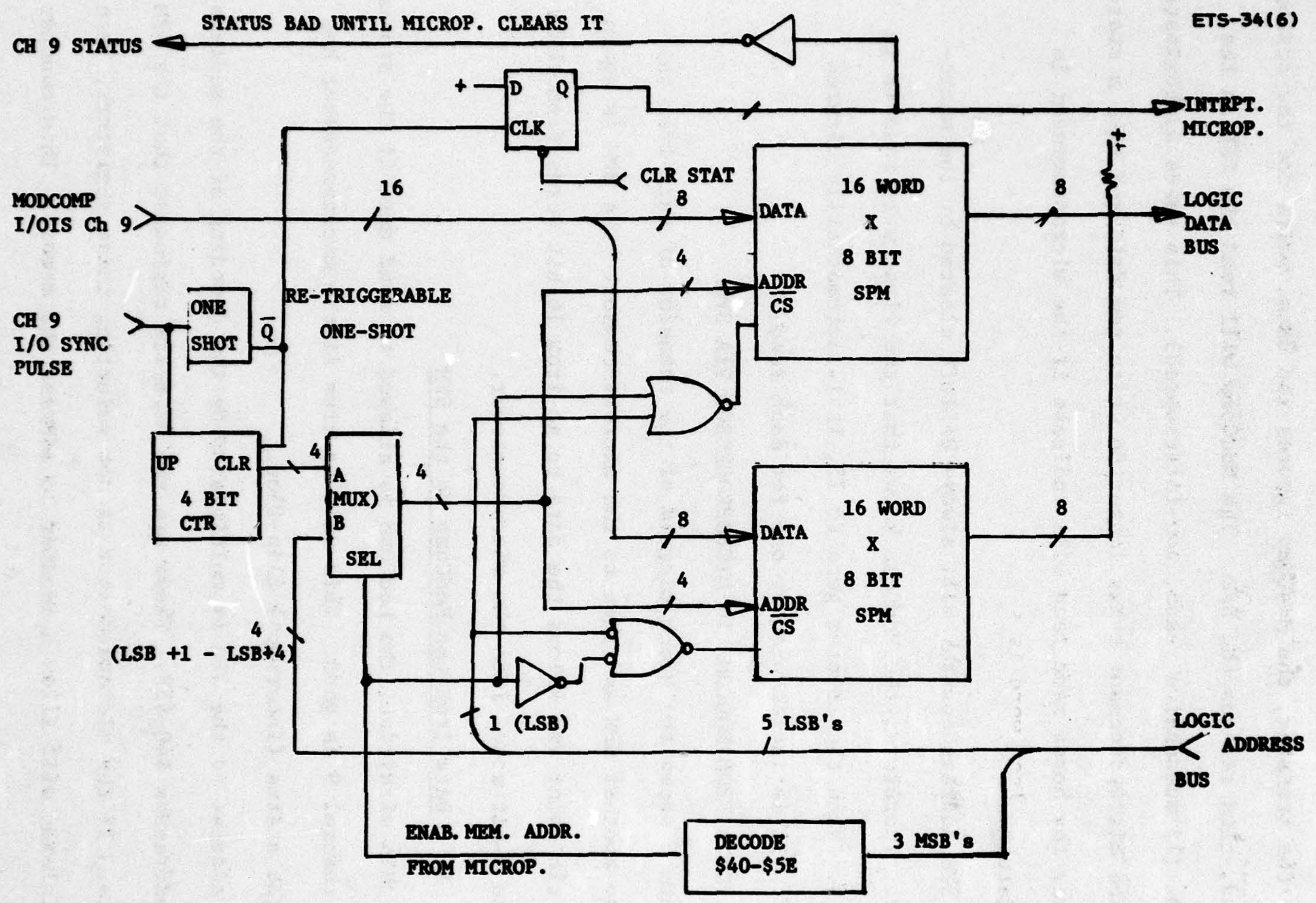


Fig. 6. Block diagram of ModComp to microprocessor — stringed message transfer — interface.

is incremented by the I/O sync pulse associated with the output data from the ModComp. The address starts at Ø because the counter is kept cleared by a re-triggerable one-shot. The clear is lifted upon the first word sent from the ModComp. The ModComp must send the data in the so-called burst mode, which means uninterrupted words at 1.6 μs apart. The one-shot has a 4 μs time out so that 4 μs after the last word is sent, the counter will clear back to Ø.

When the last word is sent, the $\overline{Q}$ side of the one-shot will set the status-interrupt flip/flop, interrupting the microprocessor and setting the ModComp status bad (=Ø).

The interrupt indicates to the microprocessor that the SPM has data to be read. After the data are read and processed, the microprocessor will clear the status flip/flop alerting the ModComp that it may send more data.

2. Data from the SPM to the Microprocessor

When the microprocessor is interrupted, it will read the data from the SPM as though the SPM represents registers \$4Ø-\$5D. Thus, the first 8-bit word will be read by the microprocessor putting a \$4Ø on the address bus. This will switch the multiplexer to steer the address to the SPM from the microprocessor address bus. The LSB of the address bus will go to the chip select of the SPM. One SPM will have this line inverted. So when register \$41 is selected, the opposite SPM will put data on the data bus. The next four bits will go to the address lines of both SPM's. These four bits will only change after every two reads by the microprocessor.

Again, the LSB of the microprocessor register address controls the chip select, thus enabling only one SPM per word read by the microprocessor. The next 4 address bits go to the address lines of the SPM and will only change after every two reads by the microprocessor.

For the telescope system at GEODSS, it has been decided that when the ModComp software has sent a burst of data, it will make the last word a #FFFF. The #FFFF indicates to the microprocessor software that the rest of the data in the SPM is not valid for this message.

After the microprocessor software detects two successive $FF's (the first being on an even register address), it knows that this is the last word sent by the ModComp. It will then process the data and finally clear the status flip/flop, indicating to the ModComp that more data can be sent. Although not specifically shown in the block diagram, the "clear status pulse" is generated by the microprocessor selecting register $5F and output anything to it. The logic address bus will no longer be sitting between $40-$5E. Thus, the "enable memory address" (signal from the decode logic) will switch the address multiplexer to the ModComp's control.

VI. SOFTWARE CONSIDERATIONS FOR THE TWO INTERFACES

A. WORD BY WORD INTERFACE

The interface between the EXORciser and the ModComp uses channels 11 and 12 on the I/OIS. ModComp to EXORciser is through channel 11. This is an output channel. EXORciser to ModComp is through channel 12 which is an input channel.

1. EXORciser to ModComp

When the EXORciser is sending data to the ModComp, it does so with interrupt handshaking. It sends an 8-bit word to logic register $Ø2, interrupting the ModComp on the I/OIS Data Interrupt 2 at vector location #A2. The status to the EXORciser is set to Ø. The status is determined by reading register $ØØ, MSB bit. It is a Ø when the data has been sent to the ModComp and goes to a 1 when the ModComp reads the data. The ModComp initializes this channel by doing a dummy read from channel 12 before enabling the interrupt. The 8 bits of data are in bits Ø8 - 15 of the ModComp, with bit Ø8 being the MSB.

2. ModComp to EXORciser

The EXORciser reads data sent by the ModComp whenever the LSB bit in hardware register $ØØ goes to a 1. This indicates that the ModComp sent a word. The EXORciser loops on testing this bit until the bit goes to a 1; then it reads the 8-bit word the ModComp has sent. The EXORciser reads this word from hardware register $Ø1. Upon reading this data, the ModComp is interrupted through Data Interrupt 1 on the I/OIS. The vector location for this interrupt is #A1. If the ModComp has more data to send, it may do so. Otherwise, it will ignore the interrupt. The status to the EXORciser stays bad until the ModComp sends another 8-bit word. The data is sent through channel 11 with the 8 bits being in bits Ø8-15, bit Ø8 being the MSB. The channel is initialized by the EXORciser doing a dummy read from register $Ø1.

B. STRINGED MESSAGE INTERFACE

This interface uses channel 8 (input to ModComp) of the I/OIS and channel 9 (output from ModComp) for transfer of data.

1. <u>Data from ModComp to Microprocessor</u>

To send data to the telescope, the ModComp software first checks the status of channel 9. If it is good (=1), then it sends data (up to 15, 16-bit words) to the microprocessor. It must send the data in the burst mode (successive outputs with interrupts disabled). The last word sent will be all 1's. After the last word is sent, the microprocessor will be interrupted, at which time it selects register $4Ø and reads data until two successive $FF's are encountered. The first $FF must be on an even address. Two successive $FF's indicate an end of transmission of data. The microprocessor processes this data; and when it is ready to accept more data, it clears the status to I/OIS channel 9 by outputting anything to register $5F. Clearing the status allows channel 9 to send more data.

2. <u>Data from Microprocessor to ModComp</u>

To receive data from the microprocessor, the ModComp reads the data from I/OIS channel 8. It does so by first checking the status. If it is good (=1), it reads 15 words in the burst mode (successive input data instructions with interrupts disabled). The status is again checked. If the status is now bad, the data is ignored. The procedure is repeated. If the status is good, the data is assumed good and is processed.

The microprocessor will send 30,8-bit words to the SPM by first selecting register $EØ. The data can be sent at any time, since the microprocessor has precedence over the interface. Thirty 8-bit words will be sent out. During this time, I/OIS channel 8's status will go bad. It will go good when address register $FE is selected. Since the register address bus is incremented after each data word is sent by the microprocessor, the software does not have to keep re-selecting a new register address.

19

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER 18 ESD-TR-79-29	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) 6 Two Ways To Interface A Motorola M6800 Microprocessor To A ModComp Computer,		5. TYPE OF REPORT & PERIOD COVERED 9 Project Report,
		6. PERFORMING ORG. REPORT NUMBER 14 ~~Project Report~~ ETS-34
7. AUTHOR(s) 10 Lawrie E. Eaton		8. CONTRACT OR GRANT NUMBER(s) 15 F19628-78-C-0002
9. PERFORMING ORGANIZATION NAME AND ADDRESS Lincoln Laboratory, M.I.T. P.O. Box 73 Lexington, MA 02173		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS Program Element No. 63428F 16 ~~Project No.~~ 2128
11. CONTROLLING OFFICE NAME AND ADDRESS Air Force Systems Command, USAF Andrews AFB Washington, DC 20331		12. REPORT DATE 11 22 February 1979
		13. NUMBER OF PAGES 34
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office) Electronic Systems Division Hanscom AFB Bedford, MA 01731 12 34p.		15. SECURITY CLASS. (of this report) Unclassified
		15a. DECLASSIFICATION DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

None

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

interfaces microprocessor ModComp

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

Two interfaces have been developed between a Motorola M6800 microprocessor and a ModComp computer. One interface uses a word by word handshaking-interrupt design to control the main telescope at the GEODSS ETS. The other interface uses a stringed message transfer for communication to the microprocessor development system. Both interfaces are described in this report.

207 650

LB